
BurgerTime: World Tour

2011 Monkey Paw Games / Konami Digital Entertainment

Position – **Lead Designer**

Development Time – 12 Months

Number of Developers – 10

Platforms – Xbox 360 Live Arcade, Playstation 3 Network, Nintendo WiiWare

Design Process

Concept-Vision Stage



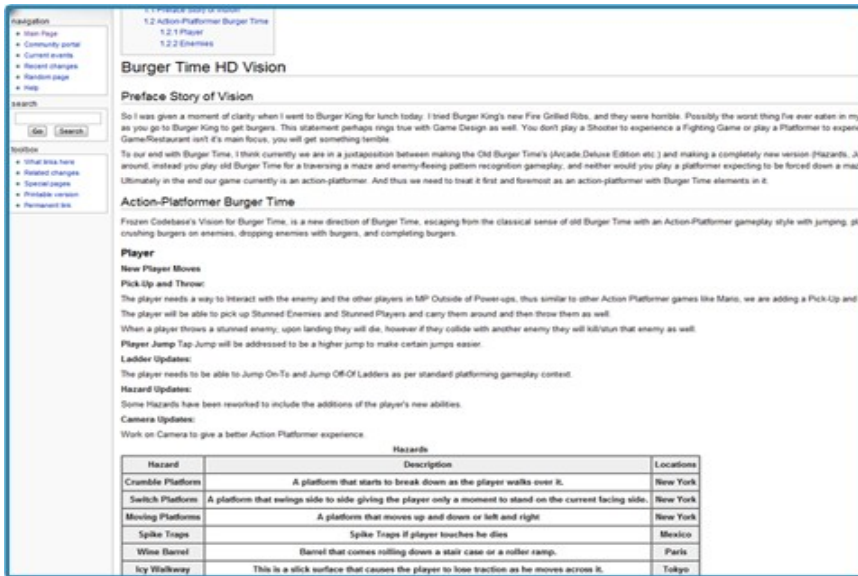
Concept

The first step in designing our game was our concept and vision stage, where our artists work with the designers to create the character concepts, the level looks, and create a mock gameplay screenshot to give a clear vision of what our game was going to be. We knew this would change and grow over the time of development, but it was important to have a clear background at the beginning stages of what we wanted to do. As a Lead Designer, I was there in the beginning for the concept and vision stages, and in phone meetings with the publisher, making sure we understood what they wanted and they understood our vision of the game.

Vision

We decided on an environment that rotates, or as we called it the Rotation Environment System. Our vision with the publisher was that of a rotating world viewed from a 2D camera perspective in a 3D world. We also decided on platformer gameplay style, so our character would be able to run and jump. We also decided our character would have powerups, this included the ability to throw pepper from the original BurgerTime arcade game. We also decided to keep with the original score-based gameplay as the player's goal to get the best score possible. We wanted to capture the old maze-like ladder gameplay feeling of the original BurgerTime, yet update it with our new vision. The goal of each level would still be the same to create each burger, but the levels would be laid out for a player with new abilities and new enemies. After this, the artist created a mock video of our gameplay vision to show what gameplay would look like in real motion.

High Level Documents



We kept all of our high level documents on an Internal Wiki. This allowed developers to quickly access documents like the General Design Document (GDD), the Technical Design Document (TDD), Vision Statements, Enemy/Player Specifications, Requirements, and more. I wrote most of these.

GDD

Here I would write things like our game flow, Enemy/Player Specifications/Unique Actions, A high level example of our worlds and levels, Boss descriptions and specifications, Controls Specification and the definitions for our score-based gameplay. I would also write Vision Statements here for people who were unclear on the path of the game.

TDD

Here we listed what was needed technically for the XBLA/PS3/WiiWare. Each version needed to be tailored specifically as Xbox 360 has different UI requirements than PS3, and leaderboards/matchmaking work differently, and with WiiWare we only had 40MB of space, so the game had to drastically change. We would also put in notes here about our engine, which collision masks and surfaces we used for certain objects, and standards for our game.

Proof of Concept



The next step we took was to quickly make a proof of concept to the publisher and create a prototype of a running game in our engine. We got a first pass version of all of our core gameplay mechanics – running, jumping, burgers dropping, enemy movement, climbing ladders, and other gameplay systems. Once we had it running we were able to give a playable version of our game vision to the publisher to show them a closer to end result.

Production

Once Production began, we began focus work on core parts of the game first. The part that received the most amount of time was the Player Jump. We needed to make sure that the Jump felt good, played well, was modular enough so if we needed to change it, it would not effect our level design and cause a cascade of problems. The other end we focused on was a new ability for the player, the ability to pick up and throw enemies and other players in Multiplayer.

Jump Focus



The Jump in our game was one of our focal points as our game is more of a platformer than the maze-like ladders gameplay of the original arcade BurgerTime. However the challenge we also had was to capture the same feeling of maze-like ladder gameplay in our new version of BurgerTime. This feature was spent the most time on, and perhaps still wasn't gotten 100% right. We focus tested the jump, tweaked it, and worked on it till we felt it was right.

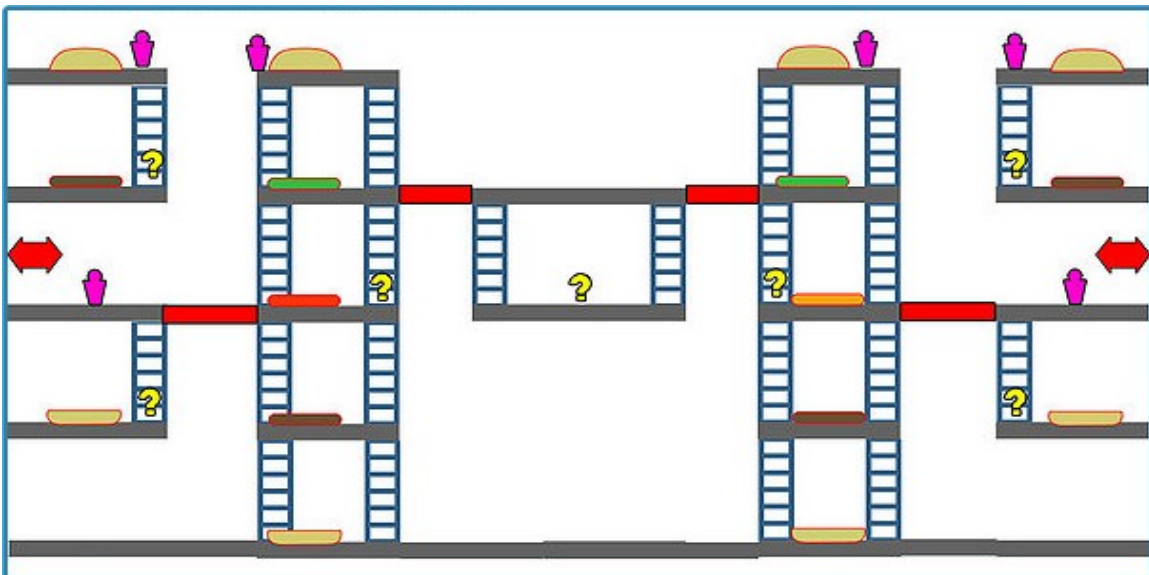
Pick-Up And Throw



One feature that came into the game when we decided the player needed more interaction abilities with the Enemies, to add more to the original BurgerTime, since our game was not just a copy of the old game but instead a brand new remake with new abilities and features.

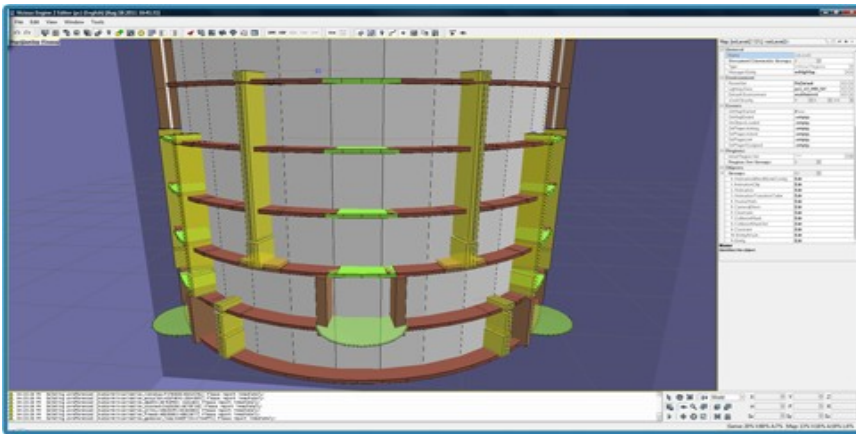
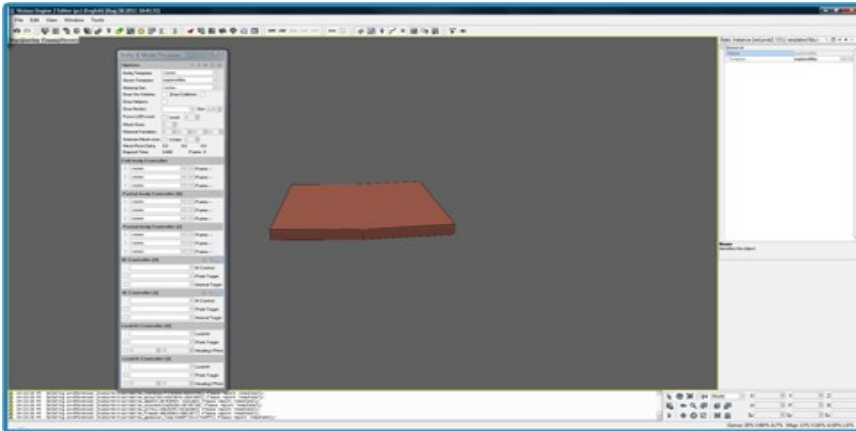
Creating a Level

Level Sketch Overview

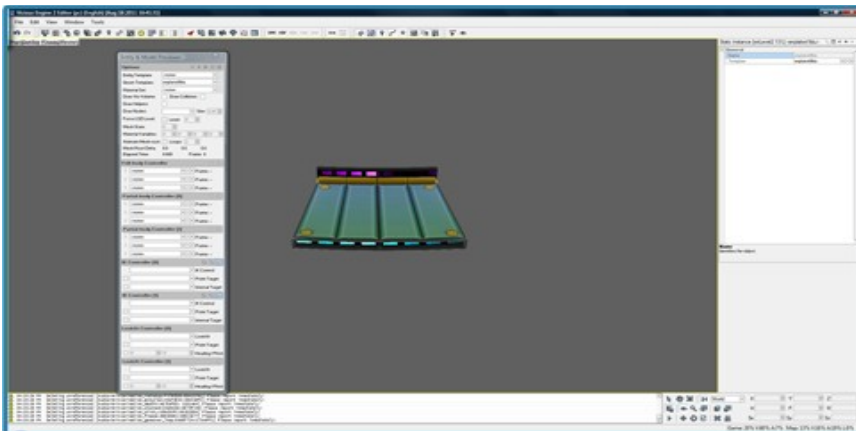


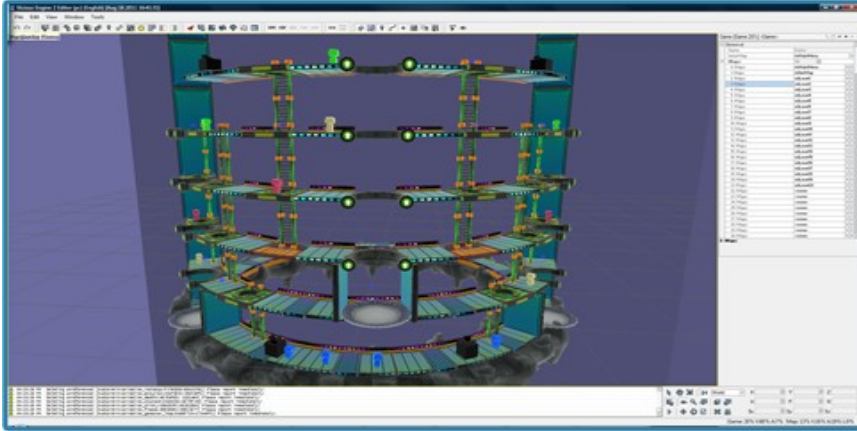
When creating the levels in BurgerTime, the first step I would do is to sketch out the level on a notebook in pen and paper. Then I would take the pen and paper sketch I did and block it out in Photoshop into a jpeg for people to see. We would lay out the level in 2D arena even though our game environment would rotate around it. This would then give us our sketch concept, which we would then use to take to the level prototyping stage and block out part of level creation.

Prototyping / Blockout Collision

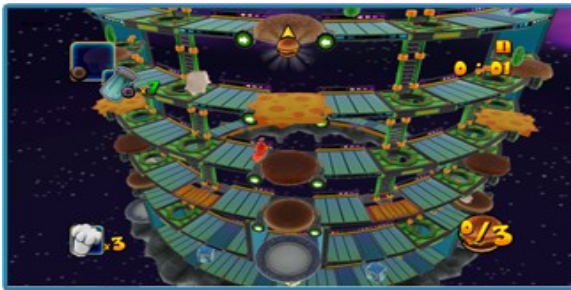


The artist would give me a workable “puzzle piece” or “building block” to build the level together with and block out the collision, and areas of the level where the player can go and the ingredients for the player to score (Burgers), and ladder collision boxes. This was called “blocking out” in our prototype phase of a level. I would put together a level with these pieces and make a playable space for the player. I put all these levels together within our game’s editor which allowed for quick iteration and testing. Next the artist would put in temp-art for me.

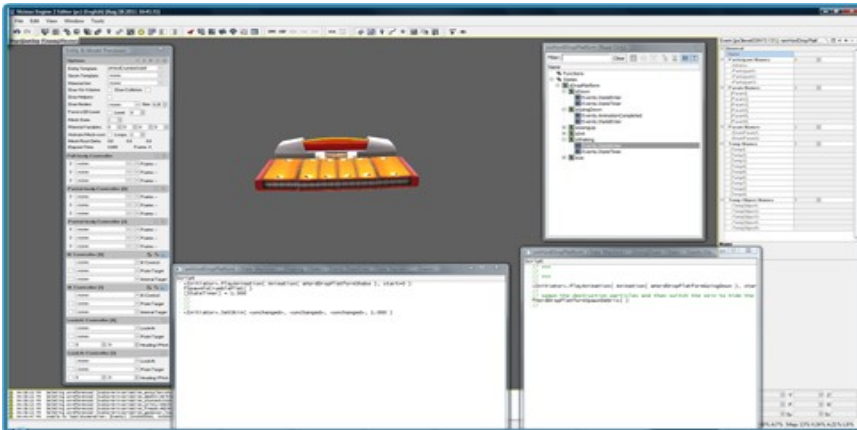




Temp-art was used for me so I could visually see parts of the level as I played. The end result in game looked like this:

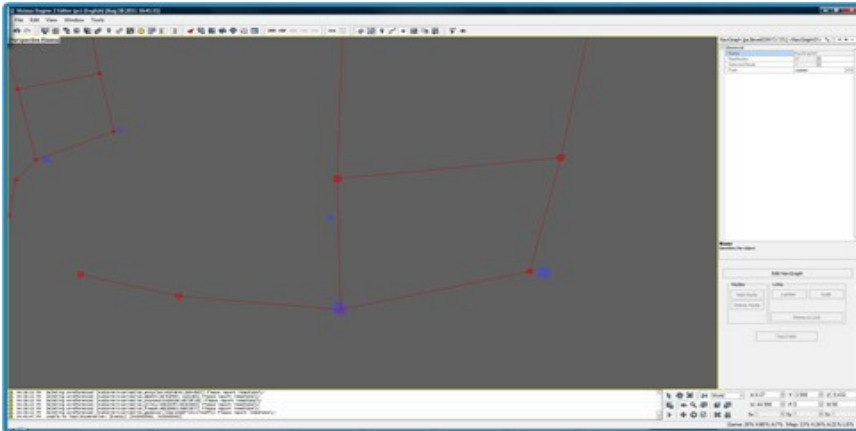


Scripting



I also scripted much of the game's environment hazards, level logic, and cinematics. Using the engine's script, finite state machine system, component and messages based entity control, I scripted in-engine a great deal almost as much as a programmer. Using the game's script I also controlled animations on objects from artists, set up hazards in levels, and other level related entities that needed scripting. In the above screenshot, an artist provided me with a "crumbling platform" where when the player steps on it, the platform explodes and crumbles away. The animations and physics debris, needed to be scripted in, so I scripted everything for the above hazard to fully work in game, like all other hazards in the game.

Enemy Placement



For our game's Enemies, we used a navigation graph, with each red box on the screenshot above representing a node, and the connecting red-lines are the edges. Using an A* algorithm, our programmer made our enemies able to navigate across these points, which I placed in each level. Certain enemies needed to act certain ways, such as the Egg enemy in our game is a guard who does not use ladders but only moves back and forth between guard points. I scripted these guard point in the games editor building off what the AI Programmer had created for me. These are the small blue boxes in the navigation graph that you see in the screenshot above.

Level Polish



After the level was finished being blocked out, the art team would come in and finalize the art, resulting in the above screenshots.

Profiling



After the level has been polished by the Art Team, and there are all full post-production effects running, and all of our lighting is complete, I need to go into each level and make sure every level is running at 30fps, and performance is top notch. If there is a dip in performance, I had to report it to the team right away, and we look to how we can solve the problem. Using in-game profiling tools I am was able to see if there were any problems on any levels, and constant play testing allowed me to spot anything wrong right away and fix it easily.

Postmortem

What we did right

For the most part I believe our team did a really great job, we put a lot of passion into this project, and made it one of our studio's best games. We spent our own time on weekends to get things we wanted into the game. We focused on gameplay first before even worrying about other technical issues. We nailed our process of creating levels early on, making it easy to prototype fun levels.

What we did wrong

The project was riddled with poor schedules, had poor time management, and miss-managed resources. This put the game on delayed schedules, and made the game come out late, missing its expected release window. Due to poor schedules, we had to rush multiplayer mode, and we wished we had spent more time on it. Because we had rushed multiplayer mode, there were many bugs that needed to get fixed later on in the QA process of the game before we shipped and almost all of them were online multiplayer related.